In the Claims

1. (Currently amended): A method of instruction execution within a microprocessor whereby:

(a) a sequence of operations from a single execution thread and across multiple basic blocks are [[is]] divided into individual strands;

(b) instructions from different basic blocks are assigned to different strands;

(c) the strands are numbered at compile time to provide an implicit logical time ordering;

~~(c)~~ (d) the operations within each individual strand are explicitly labelled with strand numbering and are executed sequentially;

~~(d)~~ (e) certain operations from different strands are executed out-of-order with respect to their original sequential order;

~~(e)~~ (f) each strand has a predication state that determines whether certain operations from the strand should be completed.

2. (original) The method according to claim 1 whereby operations in one operation strand are able to modify the predication status of another strand.

3. (original) The method according to claim 1 whereby a plurality of strands are further composed into an executable code block.

4. (original) The method according to claim 3 whereby means are provided to reset the predication status of each individual strand at the start of the execution of the code block.

5. (original) The method according to claim 4 whereby certain operation strands can be given an abort status indicating that certain operations in the block could not be completed.

6. (original) The method according to claim 5 whereby the abort status mechanism may be used to support the recovery from data speculative operations between strands.

7. (original) The method according to claim 6 whereby execution of the code block should be repeated when an abort occurs.

8. (original) The method according to claim 7 whereby the predication status of individual strands is set upon a repeat execution such that strands that have already been completed are not re-executed.

9. (original) The method according to claim 3 whereby the subdivision of code into strands is performed from an original sequential stream of instructions.

10. (original) The method according to claim 3 whereby the subdivision of code into executable blocks is performed from an original sequential stream of instructions.

11. (Previously presented) The method according to claim 3 whereby each operation strand is subdivided into a number of phases according to the type of operations that may be issued and operations that modify processor state that is visible outside of the executable block may only be executed in the final phase of each strand.

12. (Previously presented) The method according to claim 3 whereby operations from the strands may be tagged within their execution word format to indicate the strand to which they belong.

13. (Previously presented) The method according to claim 12 whereby a tagged strand number is utilized in the control logic of the functional unit to affect the execution of the operation.

14. (original) The method according to claim 13 whereby execution from a disabled strand substantially disables the operation or prevents writeback of results that could affect processor state.

15. (original) The method according to claim 14 whereby the tagging of operations may be selective and need only necessarily apply to operations that affect processor state that is visible outside of the executable block.

16. (Previously presented) The method according to claim 14 whereby an execution state of each operation strand is distributed to certain functional units by a global bus structure.

17. (original) The method according to claim 16 whereby the strand execution state is calculated and maintained in a strand control unit.

18. (original) The method according to claim 17 whereby the strand control unit receives requests to modify strand status from one or more functional units.

19. (Previously presented) The method according to claim 7 whereby an abort mechanism is utilized to provide a load speculation mechanism allowing memory loads to be executed earlier than a logically preceding store operation that may access the same address.

20. (original) The method according to claim 19 whereby the load speculation mechanism provides recovery from incorrect speculations by repeat execution of the executable block without the requirement for special compensation code.

21. (Previously presented) The method according to claim 19 whereby detection of incorrect load speculations is performed by an explicit functional unit that is used to compare the addresses being used by the load and store operations.

22. (Previously presented) The method according to claim 21 whereby address checks are inserted into the code strands as a result of insertion of such operations within a graph representation of the strand built at code generation time.

23. (Previously presented) The method according to claim 3 whereby each strand has a committed phase and entry to the committed phase of each strand is performed in the implicit logical time ordering of the strands.

24. (Previously presented) The method according to claim 23 whereby an abort of a certain operation strand also causes an abort of all strands which are logically later.

25. (Previously presented) The method according to claim 23 whereby a branch executed from a particular operation strand causes all strands which are logically later to be disabled.

26. (Previously presented) The method according to claim 25 whereby branches may be issued out of their original sequential order but are suitably resolved and no actual branch is performed until the end of the executable block is reached.

27. (Previously presented) The method according to claim 1 whereby operations from different strands may be interspersed in an execution word for the purposes of improving code density.

28. (Previously presented) The method according to claim 1 whereby an explicit operation may be issued that disables a set of strands depending on whether they are logically the first being executed.

29. (Previously presented) The method according to claim 10 whereby an operation strand mechanism is used to convert conditional blocks of code constructed using branches into separate operations strands that do not require a branch.

30. (Previously presented) The method according to claim 3 whereby the execution status of strands upon entry to the executable block may be set from a parameter provided by a preceding branch operation.

31. (Previously presented) The method according to claim 30 whereby an entry mechanism may be used to reposition a branch to a logically later strand in the block.

32. (Previously presented) The method according to claim 1 whereby the scheduling and construction of strands is influenced by profiling of the code.

33. (Previously presented) The method according to claim 32 whereby a strand ordering is used to implement static speculations that provides performance benefit whilst seeking to minimize the chances of an incorrect speculation that requires recovery.

34. (Previously presented) A microprocessor for executing instructions using the method of claim 1.